INRIA

CENTRE DE RENNES
IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 578

DESIGNING SYSTOLIC ARRAYS WITH DIASTOL

Patrice FRISON
Pierrick GACHET
Patrice QUINTON

Octobre 1986

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 - RENNES CÉDEX
FRANCE
Tél. : (99) 36.20.00
Télex : UNIRISA 95 0473 F

DESIGNING SYSTOLIC ARRAYS WITH DIASTOL
CONCEPTION D'ARCHITECTURES SYSTOLIQUES AVEC DIASTOL

Patrice **FRISON**, Pierrick **GACHET**, Patrice **QUINTON**

Publication Interne n° 314
Octobre 1986
16 pages

RESUME:

On présente une méthode de synthèse d'architectures systoliques qui permet d'aboutir à une description fonctionnelle détaillée de l'architecture utilisable pour la conception d'un circuit. Cette méthode étend la technique dite de "projection des dépendances" qui est utilisée dans le logiciel DIASTOL. La méthode est illustrée par l'exemple de la conception d'un circuit pour le produit de matrices.

ABSTRACT:

We present a method for the synthesis of systolic arrays. This method can be used for the detailed design of systolic architectures ; the result is a starting-point for the design of a VLSI chip. The method extends the so-called dependence mapping technique that was implemented in the DIASTOL system. We illustrate the method on the design of a systolic array for the multiplication of matrices.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L.A. 227)
UNIVERSITÉ DE RENNES 1 I.N.S.A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

Designing Systolic Arrays with DIASTOL(*)

Patrice FRISON, Pierrick GACHET, Patrice QUINTON

IRISA, Campus de Beaulieu,
35042 RENNES-Cedex
FRANCE

1986 Workshop on VLSI Signal Processing, IEEE, Nov. 5-7 1986, Univ. South. California, Los Angeles, USA

1. Introduction

These last few years, there has been a fastly growing interest for systolic arrays as a particular means of implementing special-purpose VLSI architectures. Domains that could most benefit from this kind of organization are signal processing and numerical analysis [1].

The DIASTOL system we present here is based on the formal method described by Quinton [2], [3]. DIASTOL is a system whose purpose is to allow systolic architectures to be designed quickly and reliably. The design process starts from the equations of the algorithm to be implemented. These equations are entered into the system and transformed until they become so-called *uniform recurrent equations* (URE). Then the synthesis program helps the designer building various systolic arrays by using the *dependence mapping* procedure. This results in an abstract (draft) description of the design, comprising the number of cells, the topology of the array, its connectivity, the timing of the data movements, and the cell fonctionnality. This abstract specification can be used as a starting point for the functional design of the cells. Depending on the type of the calculations performed by the elementary cells, DIASTOL offers several design styles based either on parallel pipelined, parallel skewed or bit-serial hardware operators. The hardware design process consists in associating consistently operators with the function symbols of the URE. The detailed timing of the design is then automatically found from the characteristics of these operators, by using a refinement of the dependence mapping procedure. A library of basic operators makes its possible to obtain quickly a functional description of a chip implementing the algorithm. The goal that is pursued is to attain designs that are specified at the bit level, as described for example by McCanny et al.[4].
This paper gives an informal overview of the operation of DIASTOL. A more formal treatment can be found in [2], [3] and [5]. In the second section, we present a simplified model of

(*) This work is partially supported by the french Research Co-ordinated Program C^3.

hardware operators that allows several common design styles to be represented. In section three, the principles of the dependence mapping procedure are briefly recalled and exemplified on the matrix multiplication algorithm. The extension of this method to the detailed design of the cells is also presented. In section four, exemples of designs for the matrix multiplication are developed.

2. A Simple Model of Hardware Operators

Our goal in this section is to give an abstract model of hardware operators, so that the timing characteristics of parallel, parallel skewed or bit-serial operators can be described within a single framework.

2.1 Presentation of the model

For the purpose of this paper, we modelize an hardware operator as a "black box" Op (figure 1) having several input ports I_i and one output O (multi-output operators are not considered here for the sake of simplicity). Each input (or output) port has $n(I_i)$ (respectively n(O)) bits. Associated with the operator is a function symbol f(Op) and several attributes that describe its timing behaviour. We assume that this timing is expressed relative to a reference time, in term of discrete time steps such that clock cycles or phases. Each operator is characterized by its type, i.e. *parallel* or *bit-serial*, and a set of timing paramaters. The *period* of the operator, denoted as $\psi(Op)$ is the time that must separate the input (resp. the output) of successive data entering the operator. The *skew* of the operator, denoted as $\sigma(Op)$ is the time that separates successive bits of the input or output data. When an operator is skewed, we assume that the data have the same skew, and that they enter the operator *least significant bit first*. Finally, to each input or output port P of the operator, is attached an *offset* denoted as $\omega(P)$, that defines at what time the first bit of the input is entered or delivered. In figure 2, several operators are schematized. Operator a) has skew 0 and period 1. The offset of its inputs are 0, and the offset of its output is 2. This represents a two stage pipeline parallel operator, with *latency* 2. Operator b) is a parallel skewed operator with the same input and output as in operator a). The bit skew is 1. Such an operator is useful in VLSI implementation in order to break delays due to the propagation of long signals (such as carries for example). Finally, operator c) is a bit-serial four bit input output operator. Its period is 4. The offset of the second input is 1, which indicates that the second input has to be delayed by one cycle relative to the first one. Notice that there may exist relationships between the parameters of an operator: for example, the period of a bit-serial operator must be a multiple of its skew.

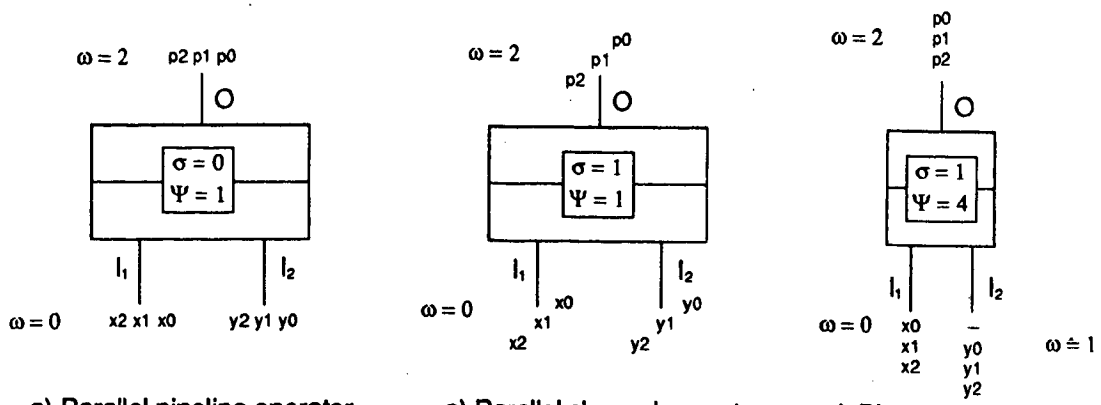


Figure 1 - Examples of operators

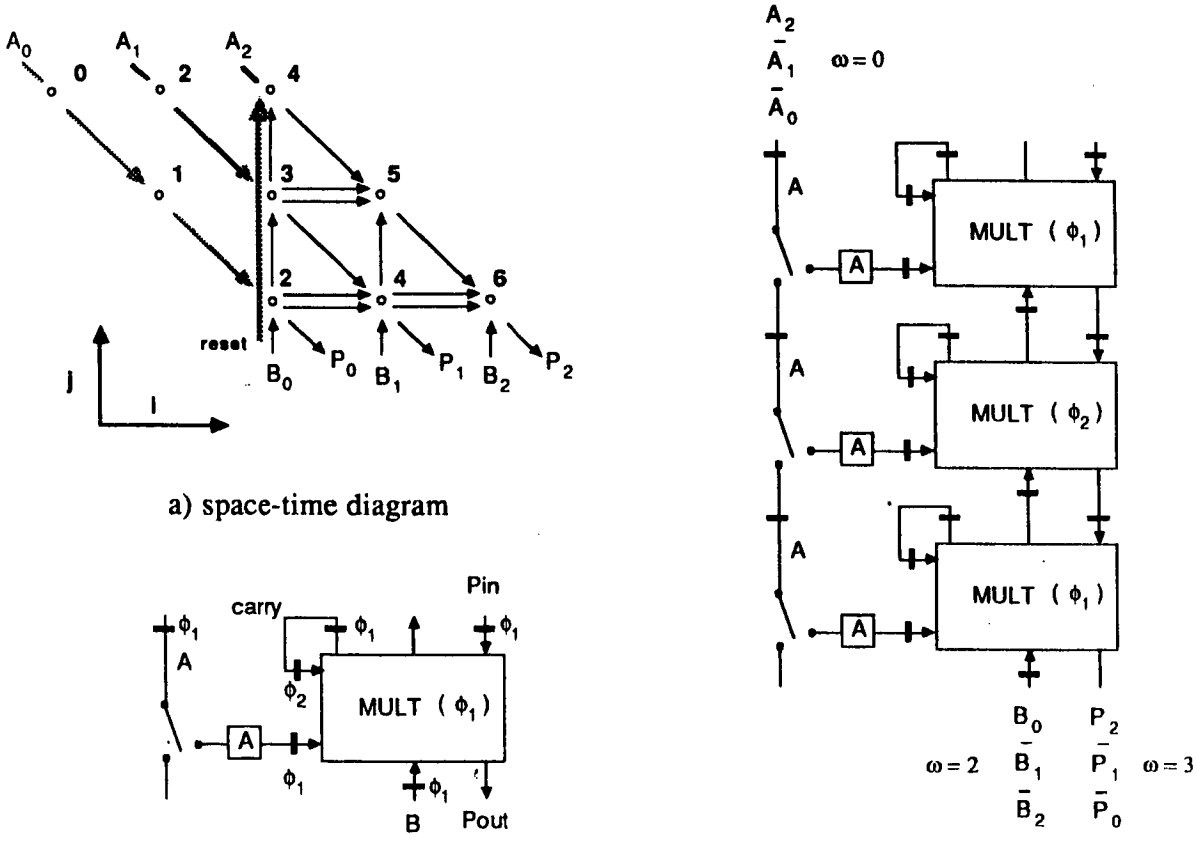


Figure 2 - A bit-serial multiplier

2.2 Examples of operators

For the purpose of this paper, we shall consider a particular case of bit-serial operators. A well known bit-serial multiplier has been described by Lyon[6]. We describe here a similar multiplier, which has the advantage of needing less latches than that of Lyon when used with a two phase non-overlapping clock. The diagram of figure 2(a) represents the time-space diagram of the multiplication of two n bit numbers $A = A_{n-1} \ldots A_0$ and $B = B_{n-1} \ldots B_0$. The result is a 2n number $P = P_{2n-1} \ldots P_0$ (truncation is not considered here). The bit-serial multiplier which corresponds to this diagram is shown on figure 2(b). Each point of the diagram corresponds to a full-adder cell. The carry bits circulate together with the bits of A. The product bits are formed along the anti-diagonal of the diagram, and leave the array by the bottom row. The shadowed arrows represent the movement of the bits of A necessary for loading this operand. A reset signal follows the bits of B and is high only when associated with B_0. It has the role of loading the bits of A, as well as setting to low the inputs P and the carries of the full adders of the line i=0. The multiplier contains n full-adder cells that work on opposite clock phases. For example, cell 0 is clocked on ϕ_1 whereas cell 1 is clocked on ϕ_2. The partial product bits flow from the top to the bottom, as well as the bits of A. The bits of B and the reset bits flow from the bottom to the top. The exact timing of this operator is best described in figure 2(a). One can see that the operands have skew 2. The period of the operator is 2n. Finally, the offset of A, B and P are respectively 0, n-1 and n. The main difference between this design and that of Lyon is that the B bits and the reset bits do not need being latched during one full clock cycle between consecutive full adder cells, which saves four latches for each cell. On the other hand, it is fair to recognize that Lyon's design has a simpler timing, as A and B would have the same offset. Notice that our design can be extended easily to handle two's complement binary numbers by using Booth recoding algorithm.

A straightforward bit-serial adder for two 2n bit binary numbers is shown in figure 3. It has period 2n and skew 2. The offset of the inputs is 0, and the offset of the output is 1.

3. Detailed design of systolic arrays using dependence mapping

In this section, we describe the principles of the DIASTOL system, and we show how the basic method can be extended to handle the design of the cells.

3.1. Introduction

Consider the matrix multiplication of two NxN matrices C=AB. The coefficients of C are given

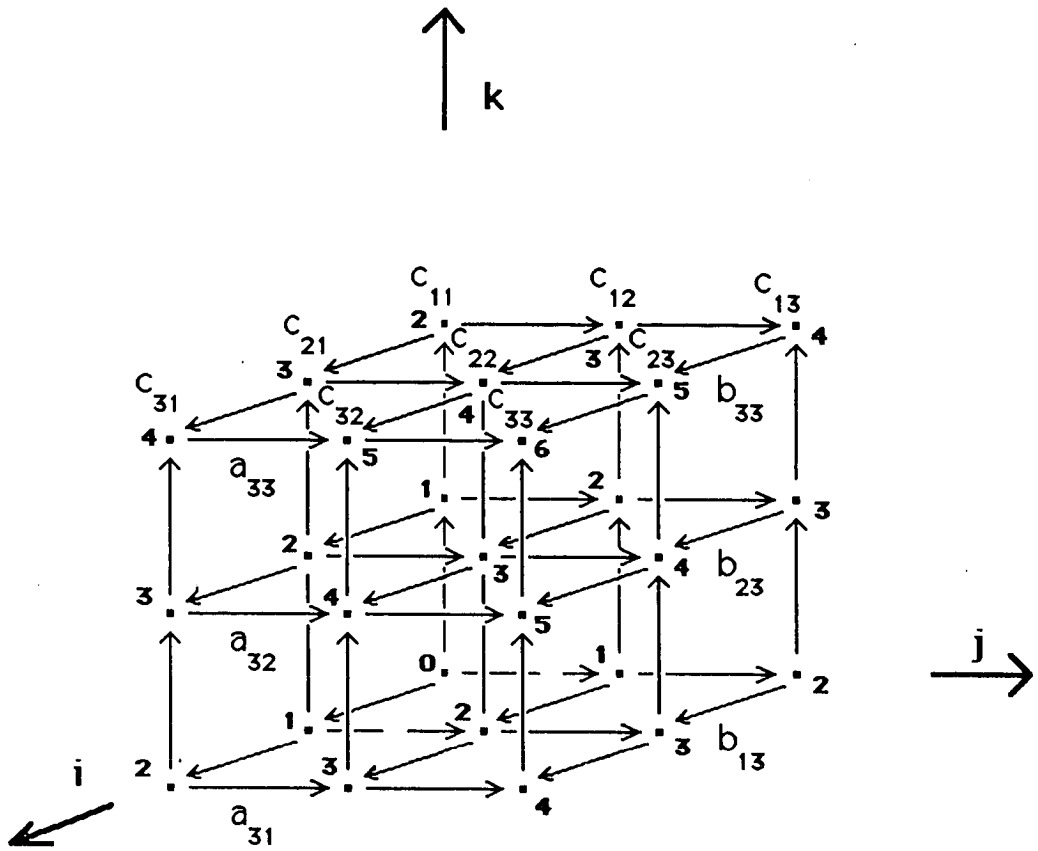


Figure 4 - Dependence graph for the matrix multiplication

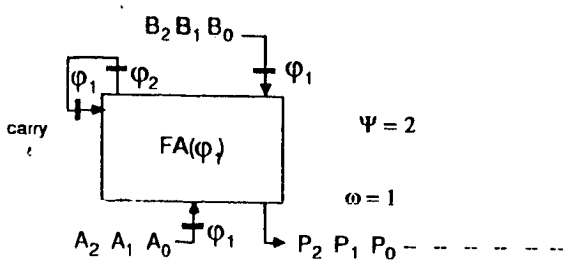


Figure 3 - A bit-serial adder

by the following equation :

$$\forall\, i,j : 1 \le i \le N,\ 1 \le j \le N,\ c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj} \qquad (1)$$

The fist step of the method consists in rewriting (1) as a left-to-right summation, in order to avoid the use of unbounded arity operators. This gives

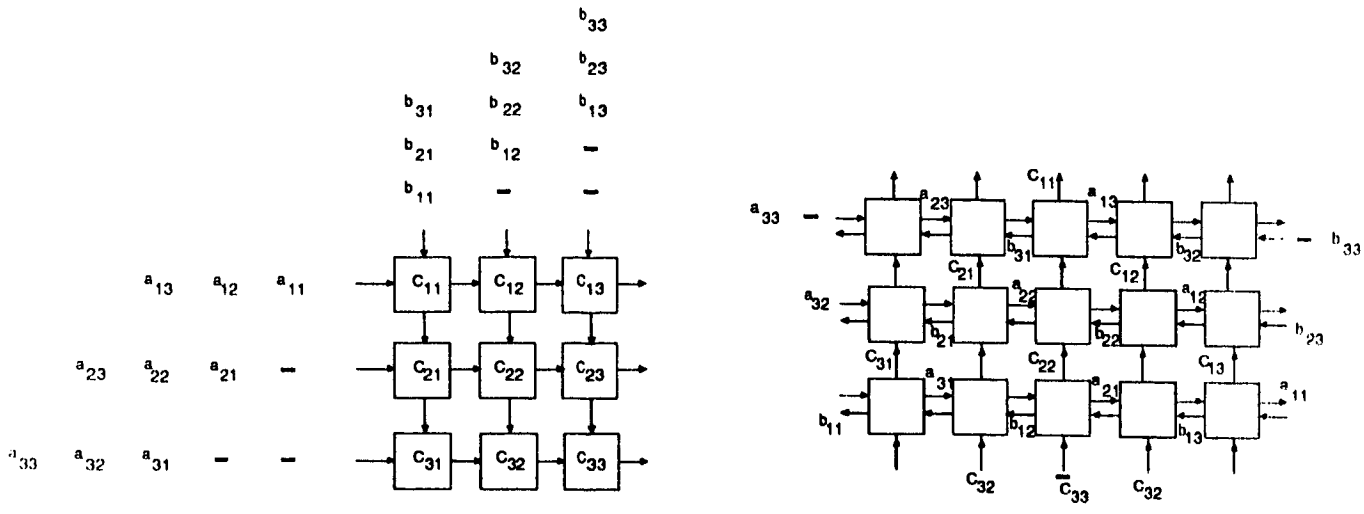
$$\begin{aligned} &\forall\, i,j,k : 1 \le i \le N,\ 1 \le j \le N,\ 1 \le k \le N, \\ &\quad C(i,j,k) = \text{if } k = 1 \text{ then } 0 \\ &\qquad\qquad \text{if } k \neq 1 \text{ then } C(i,j,k-1) + a_{ik} b_{kj} \\ &\quad c_{ij} \equiv C(i,j,N) \end{aligned} \qquad (2)$$

In this new equation, c_{ij} is obtained as the final value C(i,j,N) of the accumulation. The whole computation can be represented as a set of elementary multiply and add calculations, each one of which being associated with an integral point of a cube of size N (figure 4). The set of points that are considered is called the *domain of computation*, denoted as D in what follows. In order to obtain a design without broadcasting a and b, it is necessary to pipeline the values a_{ik} and b_{kj}. These values are replaced by new variables A and B, and two new equations which represent the circulation of these variables are added, i.e. :

$$\begin{aligned} &\forall\, i,j,k : 1 \le i \le N,\ 1 \le j \le N,\ 1 \le k \le N, \\ &\quad C(i,j,k) = \text{if } k = 1 \text{ then } 0 \\ &\qquad\qquad \text{if } k \neq 1 \text{ then } C(i,j,k-1) + A(i,j,k) \times B(i,j,k) \\ &\quad A(i,j,k) = \text{if } j = 1 \text{ then } a_{ik} \\ &\qquad\qquad \text{if } j \neq 1 \text{ then } A(i,j-1,k) \\ &\quad B(i,j,k) = \text{if } i = 1 \text{ then } b_{kj} \\ &\qquad\qquad \text{if } j \neq 1 \text{ then } B(i-1,j,k) \\ &\quad c_{ij} \equiv C(i,j,N) \end{aligned} \qquad (3)$$

Equation (3) is a special form of recurrence equations called *uniform recurrence equation* (URE) [7] as a computation depends only of neighbouring points, in a uniform manner. In our case, the computation at point (i,j,k) depends on points (i,j,k−1), (i,j−1,k), and (i−1,j,k). Figure 4 depicts the *dependence graph* associated with equation (3).

The second step of the method consists in ordering the points of D consistently with respect to the dependences between the computations. In a first approximation, we assume that every calculation takes one unit of time. Let us denote as t(i,j,k) the time at which computation at point (i,j,k) can be done. The function t is a mapping from D to the set Z of integers, and is called the



a) Projection along the k axis

b) Projection along (1,1,0)

Figure 5 - Two different systolic arrays for matrix multiplication

timing function. If we restrict t to be an affine mapping, i.e. $t(i,j,k) = \lambda_1 i + \lambda_2 j + \lambda_3 k + \alpha$, where λ_1, λ_2, λ_3, and α belong to Z, we can see that $t(i,j,k) = i + j + k - 3$ is a possible timing function for the dependence graph of figure 4. In [2], it is shown how the timing-function can be computed automatically.

The final step of the method consists in mapping the dependence graph on a finite set of processing elements. We call *allocation function*, and denote as a(i,j,k) the mapping from Z^3 to itself that define the number of the processor where computation at point (i,j,k) is done. The simplest way is to use a projection of the space R^3. Depending of the direction of this projection, data will move or not in the final design. Consider first the projection of the cube of figure 4 along the k axis. The corresponding design is depicted in figure 5a). A given processor P_{ij} carries out the computations associated with points of segments parallel to the k axis. Therefore, c_{ij} is computed on P_{ij}. The coefficients of matrix A move from the left to the right, and those of B from the top to the bottom. Other well-known designs for the matrix multiplication can be obtained by using other directions of projection. Figure 5(b) shows a design obtained using the projection along vector (1,1,0). In this design, we can see that the c_{ij}'s move from bottom to top, whereas the a_{ik}'s and b_{kj}'s flow respectively from left to right and right to left. Notice that in this design, the cells are working only every other cycle. This is clearly seen on the dependence graph, as computations mapped on the same processor are separated by two cycles instead of one as in the design of figure 5(a).

3.3. Generalization of the method

The previous section described informally the dependence mapping procedure. This method can be use to produce quickly "drafts" of systolic arrays. In this section, we shall deal with the detailed design of the cells, taking into account the hardware technological aspects of the design of a chip. The approach taken in DIASTOL makes use of a set of predefined hardware operators, which are modelized using the parameters defined in section 2. Such operators are used to replace corresponding function symbols in the equations. The timing and allocation functions that have been defined in section 3.1. are generalized in order to allow the delays introduced by the operators as well as the timing of the data (at the bit level) to be computed automatically. Depending on the complexity of the functions in the URE, several styles of operators can be chosen. Here, we shall consider only fully parallel or bit-serial operators, as those described in section 2. Other examples can be found in [5].

Let us return to the URE of equation (3). In order to separate the multiply and plus functions

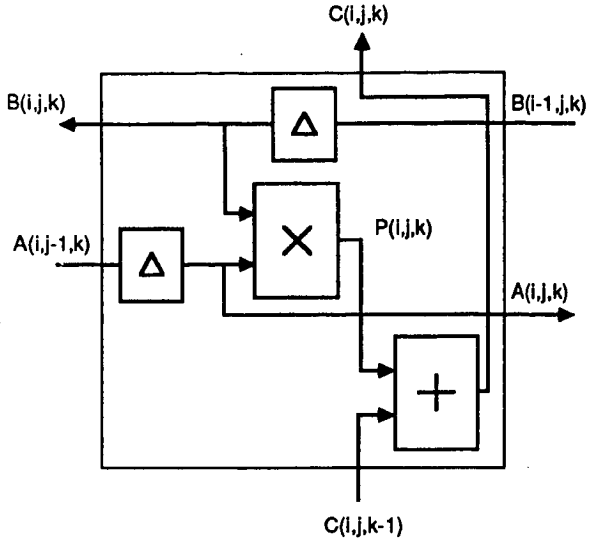


Figure 6 - Virtual mapping of the cell for matrix multiplication

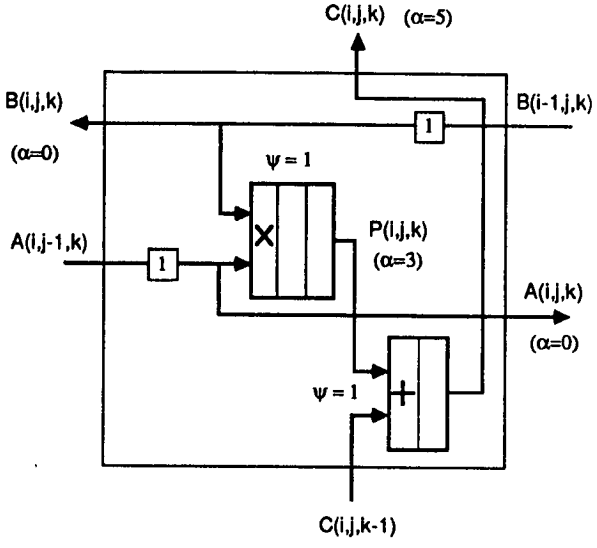


Figure 7 - Mapping of the cell when using a 3 stage pipeline parallel multiplier and a 2 stage pipeline adder

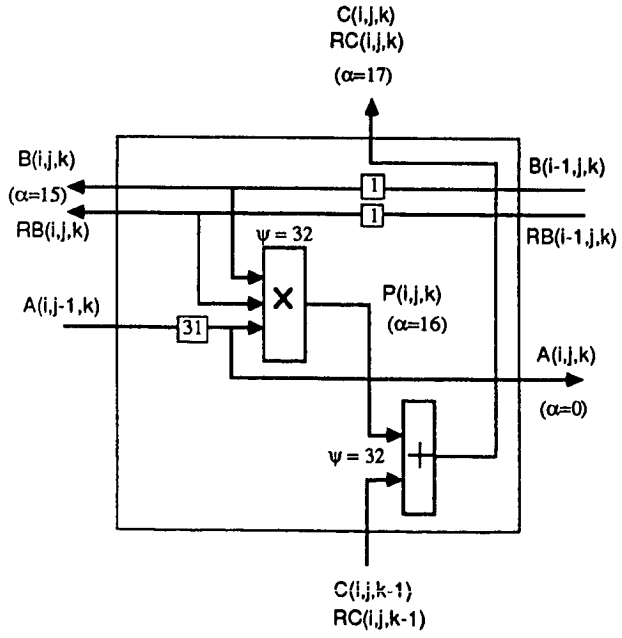


Figure 8 - Mapping of the cells when using the bit-serial operators of figure 3 and 4

defining C(i,j,k), we split the corresponding equation by adding a new variable P. This gives :

$$
\begin{aligned}
&\forall\ i,j,k : 1 \le i \le N,\ 1 \le j \le N,\ 1 \le k \le N, \\
&\quad C(i,j,k) = \text{if } k = 1 \text{ then } 0 \\
&\qquad\qquad \text{if } k \neq 1 \text{ then } C(i,j,k-1) + P(i,j,k) \\
&\quad P(i,j,k) = A(i,j,k) \times B(i,j,k) \\
&\quad A(i,j,k) = \text{if } j = 1 \text{ then } a_{ik} \\
&\qquad\qquad \text{if } j \neq 1 \text{ then } A(i,j-1,k) \\
&\quad B(i,j,k) = \text{if } i = 1 \text{ then } b_{kj} \\
&\qquad\qquad \text{if } j \neq 1 \text{ then } B(i-1,j,k) \\
&\quad c_{ij} \equiv C(i,j,N)
\end{aligned}
\qquad (4)
$$

We now attach one hardware operator to each equation (except of course to the ≡ statement), in such a way that the timing parameters of all the operators are defined relative to the same unit, and that all have the same skew σ. To an identity equation, we attach a delay in order to avoid the possible broadcasting of a data. A delay has obviously a period 1, and the offset of its input and ouptut are respectively 0 and 1. Notice however that our method still works if no delay is attached to the identity. It may then result is what Kung[8] calls semi-systolic arrays. This intermediate step is called a *virtual mapping* (figure 6) of the computations. In order to obtain the *physical mapping* that we seek, we must schedule the computations by means of a timing function and map them on processing cells. By doing so, extra delays will be introduced automatically between the operators, as we shall see in the following.

In order to schedule the computations, we associate with each variable V of the equations an affine timing function $t(i,j,k) = \lambda_1 i + \lambda_2 j + \lambda_3 k + \alpha_V$, that gives the time (in term of the common unit of the operators) at which *the first bit* of V(i,j,k) is computed. It must be pointed out that λ_1, λ_2, and λ_3 are kept independent of V. In other words, all the variables will be computed at the same rate, possibly with a slight constant time offset which depends on the relative values of the scalars α_V. Let us consider simultaneously the timing function and the allocation function. Consider the projection along the vector u = (0, 0, 1), which gives the design of figure 5(a). The parameter to be determined are therefore λ_1, λ_2, λ_3, α_C, α_A, α_B, and α_P. Consider the equation C(i,j,k) = C(i,j,k−1) + P(i,j,k). Assume that we associate with this equation an adder with period ψ and skew σ, and let ω(O), $\omega(I_C)$, and $\omega(I_P)$ be the offset associated respectively to the output and the inputs C and P of the adder. Clearly C(i,j,k) cannot be computed before the time at which C(i,j,k−1) is available augmented by the time needed by the operator to produce C(i,j,k−1), namely $\omega(O) - \omega(I_C)$. Therefore, we must have :

$$t_C(i,j,k) - t_C(i,j,k-1) = \lambda_3 \ge \omega(O) - \omega(I_C)$$

Applying a similar reasoning to C(i,j,k) and P(i,j,k) gives :

$$t_C(i,j,k) - t_P(i,j,k-1) = \alpha_C - \alpha_P \geq \omega(O) - \omega(I_P)$$

Such constraints can be obtained for each equation. They are called *latency constraints*. Another type of constraints involve the period of the operators and the direction of projection. Consider two computations that are mapped on the same processing cell. Assume that these computations are associated with points z_1 and z_2 of D. As a given equation is mapped on the same operator, the interval of time that separates the computation of this equation for successive points must be greater than or equal to the period of the operator. Consider again the case of C(i,j,k). As C(i,j,k) and C(i,j,k+1) are computed by the same operator (since u = (0,0,1)), we must have :

$$t_C(i,j,k+1) - t_C(i,j,k) \geq \psi$$

or equivalently, $\lambda^T u \geq \psi$ where $\lambda^T u$ denotes the dot product of λ and u. As the same condition holds for all the operators, we finally have the constraint :

$$\lambda^T u \geq \text{Max } \{\psi\}$$

which is called the *period constraint*. Since the timing-functions are linear, both period and latency constraints result in a system of linear inequalities. By solving the corresponding linear program, one can find admissible values of the parameters. Once the parameters are chosen, it is possible to compute the number of delays to be inserted between operators. As an example, consider the case of the delay to be inserted between the output P(i,j,k) of the multiplier and the corresponding input of the adder in the matrix multiplication cell. P(i,j,k) is available at time $t_P(i,j,k)$. On the other hand, as the result of the adder must be produced at time $t_C(i,j,k)$, its input P(i,j,k) must arrive at time $t_C(i,j,k) - (\omega(C) - \omega(P))$. Therefore, the number Δ of delays to be inserted is :

$$\Delta = t_C(i,j,k) - (\omega(C) - \omega(P)) - t_P(i,j,k) = \alpha_C - \alpha_P - (\omega(C) - \omega(P))$$

Moreover, one can notice that the number of delays (and consequently the total number of delays in a cell) is a linear function of the parameters. Therefore, in order to minimize this number, it suffices to solve the corresponding integer programming problem. It should be pointed out that this method is close to that proposed recently by Goossens et al.[9] for the delay management problem in bit-serial architectures. It can however be applied to other design styles, and embeds in a single framework the calculation of the systolic organization and the

delay management problem.

4 Application to the matrix multiplication

In this section, we describe two implementations of the matrix multiplication systolic array, using either purely parallel operators, or bit serial operators.

4.1. Parallel pipelined operators

Consider first that the multiply operator is a three stage pipelined operator (input offset = 0, output offset = 3, and period $\psi_X = 1$), and that the adder is a two stage pipelined operator (input offset = 0, output offset = 2, and period $\psi_X = 1$). The delays associated with the identity are assumed to be parallel registers. Consider the design resulting of a projection along vector u = (1,1,0), as sketched in figure 5(b). The latency constraints given by equation (3) are :

From C(i,j,k) : $\lambda_3 \geq \omega(C) = 2$

$\alpha_C - \alpha_P \geq \omega(C) = 2$

From P(i,j,k) : $\alpha_P - \alpha_A \geq \omega(P) = 3$

$\alpha_P - \alpha_B \geq \omega(P) = 3$

From A(i,j,k) : $\lambda_2 \geq 1$

From B(i,j,k) : $\lambda_1 \geq 1$

On the other hand the periodicity constraint is $\lambda_1 + \lambda_2 \geq 1$. Assuming that α_C, α_P, α_A, and α_B are non-negative integers, the convex set defined by the above constraints has one vertex given by $\lambda_1 = 1$; $\lambda_2 = 1$; $\lambda_3 = 2$; $\alpha_C = 5$; $\alpha_P = 3$; $\alpha_A = \alpha_B = 0$. This solution is therefore optimal with respect to the number of delays. The corresponding systolic array is depicted in figure 7.

4.2. Bit-serial operators

Let us implement the same systolic array using the bit-serial multiplier and adder described in section 2.2. We assume that the numbers have 16 bits. Therefore, the period of both adder and multiplier is 32. For the multiplier, the offset of A is 0, the offset of B and of the reset is n-1 = 15, and the offset of P is n = 16. On the other hand, the offset of the inputs of the adder is 0, and that of the output is 1. Finally, the skew of both operators is 2. In order to implement the systolic array using these operators, it is necessary to modify the equations so that the reset signals appear. Equation (4) is rewritten as follows, after introducing a reset signal rc_{ij} that

follows c_{ij} and a reset signal rb_{kj} associated with b_{kj} :

$$
\begin{aligned}
C(i,j,k) &= \text{if } k = 1 \text{ then } 0 \\
&\quad \text{if } k \neq 1 \text{ then } \textit{Add}\ [C(i,j,k-1)\ ,\ P(i,j,k)\ ,\ RC(i,j,k-1)] \\
RC(i,j,k) &= \text{if } k = 1 \text{ then } rc_{ij} \\
&\quad \text{if } k \neq 1\ RC(i,j,k-1) \\
P(i,j,k) &= \textit{Mult}\ [A(i,j,k)\ ,\ B(i,j,k)\ ,\ RB(i,j,k)] \\
A(i,j,k) &= \text{if } j = 1 \text{ then } a_{ik} \\
&\quad \text{if } j \neq 1 \text{ then } A(i,j-1,k) \\
B(i,j,k) &= \text{if } i = 1 \text{ then } b_{kj} \\
&\quad \text{if } j \neq 1 \text{ then } B(i-1,j,k) \\
RB(i,j,k) &= \text{if } i = 1 \text{ then } rb_{kj} \\
&\quad \text{if } j \neq 1 \text{ then } RB(i-1,j,k) \\
c_{ij} &\equiv C(i,j,N)
\end{aligned}
\qquad (5)
$$

Applying the method gives the following latency constraints :

$$
\begin{aligned}
&\text{From } C(i,j,k): && \lambda_3 \geq \omega(C) = 1 \\
& && \alpha_C - \alpha_P \geq \omega(C) = 1 \\
& && \lambda_3 + \alpha_C - \alpha_{RC} \geq \omega(C) = 1 \\
&\text{From } RC(i,j,k): && \lambda_3 \geq 1 \\
&\text{From } P(i,j,k): && \alpha_P - \alpha_A \geq \omega(P) - \omega(A) = 16 \\
& && \alpha_P - \alpha_B \geq \omega(P) - \omega(B) = 1 \\
& && \alpha_P - \alpha_{RB} \geq \omega(P) - \omega(RB) = 1 \\
&\text{From } A(i,j,k): && \lambda_2 \geq 1 \\
&\text{From } B(i,j,k): && \lambda_1 \geq 1 \\
&\text{From } RB(i,j,k): && \lambda_1 \geq 1
\end{aligned}
$$

On the other hand, the periodicity constraint is $\lambda_1 + \lambda_2 \geq 32$. Again assuming that the α's are non-negative, the convex set defined by the above inequalities has the following vertices :

$$\lambda_3 = 1$$

$$(\lambda_1 = 1 \text{ and } \lambda_2 = 31) \text{ or } (\lambda_1 = 31 \text{ and } \lambda_2 = 1)$$

$$\alpha_C = 17;\ \alpha_{RC} = 0 \text{ or } 17;\ \alpha_P = 16;\ \alpha_A = 0;\ \alpha_B = 0 \text{ or } 16;\ \alpha_{RB} = 0 \text{ or } 15$$

In order to minimize the number of delays, it is clear that $\alpha_{RC} = 16$, $\alpha_{RC} = 17$, and $\alpha_{RB} = 15$

must be chosen. Therefore, the number of extra delays is given by the expression :

$$(\lambda_2 - 1) + 2\,(\lambda_1 - 1)$$

It can be seen that the best solution is obtained when $\lambda_2 = 31$ and $\lambda_1 = 1$, which results in the design of figure 8. The intuitive explanation of this fact is simply that the design is not symmetric in A and B, as the reset signal follows B. Therefore, it is better to minimize the number of delays along the B path.

5. Conclusion

We have described the principle of the DIASTOL system. Its purpose is to allow systolic arrays to be designed at the functional level. The method underlying DIASTOL is based on the dependence mapping procedure. The detailed design of the cells is made from a library of operators that are modelized in terms of simple timing parameters called the period, the skew and the offset of the inputs and outputs. It has been shown that a refinement of the dependence mapping makes it possible to obtain automatically the exact timing of the data, and to compute automatically the number of delays to be inserted. Moreover, the number of delays can be optimized using integer linear programming. A first version of DIASTOL implementing the first step of the dependence mapping procedure has been implemented[10]. We are currently implementing the refinement step.

5. References

[1] S. Y. Kung, "On Supercomputing with Systolic / Wavefront Array Processors," *Proceeding IEEE*, July 1984, pp. 867 - 884.

[2] P. Quinton, The Systematic Design of Systolic Arrays, IRISA Research Report 193, 1983.

[3] P. Quinton, "Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations," in *Proc. 11th Annual Intern. Symp. Computer Architecture,* IEEE, June 1984, Ann Arbor, pp. 208 - 214.

[4] J. V. McCanny, J. G. McWhirter, K.W. Wood, "Optimised Bit Level Systolic Array for Convolution," *IEE Proc. F,* 131, pp. 632 - 637, 1984.

[5] P. Quinton, P. Gachet, "Automatic Design of Systolic Chips," in *Future Trends in Computing*, F. Robert and C. Di Crescenzo (eds), Masson - Wiley, 1985.

[6] R. F. Lyon, "Two's Complement Pipeline Multipliers," *IEEE Trans. Comm.*, Vol. COM-24, pp. 418 - 425, Apr. 1976.

[7] R. M. Karp R. E. Miller, S. Winograd, "The Organization of Computations for Uniform Recurrence Equations," *JACM*, Vol 14, No 3, july 1967, pp. 563 - 590.

[8] H. T. Kung, "Why Systolic Architectures ? ," *IEEE Computer*, 15, pp. 37 - 46, Jan. 1982.

[9] G. Goossens, R. Jain, J. Vandewalle, H. De Man, "An Optimal and Flexible Delay Management Technique for VLSI," in *Computational and Combinatorial Methods in System Theory*, C. I. Byrnes and A. Lindquist (eds), Elsevier Science Publishers B. V. (North-Holland), 1986.

[10] P. Quinton, P. Gachet, DIASTOL user's Manual : Preliminary version, IRISA Research Report, Aug. 1984.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique